VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Ovládání RGB LED panelu pomocí Raspberry-Pi

Controlling RGB LED Panel with Raspberry-Pi

2015 Tomáš Štrbačka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Štrbačka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Ovládání RGB LED panelu pomocí Raspberry-PI
Controlling RGB LED Panel with Raspberry-PI

Zásady pro vypracování:

Navrhněte univerzální rozhraní pro RGB LED panel založené na počítači Raspberry-PI a jeho integrovaném SPI rozhraní. Nad tímto rozhraním realizujte zobrazování standardních datových formátů, jako jsou obrázky JPEG, BMP, GIF a videa ve formátech MPEG2 a H264.

1. Popište LED panel a princip ovládání jednotlivých RGB LED pomocí multiplexovaného SPI rozhraní.
2. Vytvořte program pro Raspberry-PI, kterým budete moci na LED panelu zobrazovat testovací obrazce.
3. Navrhněte řízení panelu pomocí počítače Raspberry-PI. Nad navrženým rozhraním naprogramujte zobrazování multimediálních dat.
4. Vytvořte animace které budou propagovat tento projekt.

Seznam doporučené odborné literatury:

[1] Luděk SKOČOVSKÝ, a Scott JERNIGAN. Linux: dokumentační projekt. 4., aktualiz. vyd. Překlad Ivo Fořt, David Krásenský. Brno: Computer Press, 2007, 1334 s. ISBN 978-80-251-1525-1
[2] Eben UPTON, a Gareth HALFACREE. Raspberry Pi user guide. 2. aktualiz. vyd. Chichester, England: John Wiley, c2012. xiv, 248 p. ISBN 11-184-6446-X

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Seidl, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 28. května 2015 ..... Tomáš Šlahůčka .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28. května 2015 ..... Tomáš Šlahůčka .....

Rád bych na tomto místě poděkoval mé rodině, která mi poskytla zázemí a podporu, dále pak všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Jmenovitě bych rád poděkoval panu Ing. Davidu Seidlovi, Ph.D. a panu Ing. Petru Olivkovi, Ph.D. za cenné rady, trpělivost a zázemí, které nám ve škole poskytli.

Abstrakt

Cílem mé bakalářské práce bylo navrhnout a implementovat univerzální rozhraní pro RGB LED panel, založené na počítači Raspberry Pi, a jeho integrovaném rozhraní SPI. Tato práce popisuje LED panel, jeho ovládání pomocí multiplexovaného rozhraní SPI a způsob kterým jsem vybral pro zobrazování standardních datových formátů.

Klíčová slova: Raspberry Pi ,SPI, LED panel, OpenCV, Xlib, Framebuffer

Abstract

The aim of my thesis was to design and implement a universal interface for RGB LED board, using computer Raspberry Pi, and his integrated SPI interface. This paper describes the LED board, control it using multiplexed SPI interface and the way we've chosen to display standard data formats.

Keywords: Raspberry Pi ,SPI, LED board, OpenCV, Xlib, Framebuffer

Seznam použitých zkratek a symbolů

SPI	–	Serial Peripheral Interface
CLK,CK	–	Clock
SD	–	Serial Data
LED	–	Light-Emitting Diode
RGB	–	Red Blue Green
BGR	–	Green Blue Red
PNG	–	Portable Network Graphics
MPEG	–	Moving Picture Experts Group
IO	–	Integrovaný Obvod
GPIO	–	General-Purpose Input/Output
RAM	–	Random-Access Memory
UART	–	Universal Asynchronous Receiver/Transmitter
MOSI	–	Master Out Slave In
MISO	–	Master In Slave Out
CS	–	Chip Select
LAN	–	Local Area Network
HDMI	–	High-Definition Multimedia Interface

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době je reklama všude tam, kam se jen podíváme. Někteří lidé ji obdivují, někteří zase nesnášejí. Pravdou ale zůstává, že je velice důležitá. Všichni obchodníci stále hledají nové způsoby, jak upozornit na své produkty nebo služby. Poslední dobou začínají nahrazovat ve městech klasické veliké plakátovací plochy právě reklamní LED panely.

A tak tedy v rámci propagace Fakulty elektroniky a informatiky, na dny otevřených dveří ve škole a různé další příležitosti se rozhodl pan Ing. David Seidl, Ph.D. a pan Ing. Petr Olivka, Ph.D. sestrojit přenosnou verzi reklamního panelu.

Cílem mé práce je popsat tento LED panel, princip ovládání jednotlivých pixelu pomocí multiplexovaného rozhraní SPI, sestrojit a ověřit funkčnost hardwarového modulu jehož princip navrhl pan Ing. David Seidl, Ph.D., dále vytvoření programu pro zobrazování multimediálních dat na panelu pomocí tohoto modulu a jednodeskového počítače Raspberry Pi.

2 Popis LED panelu a principu ovládání

V této kapitole je popsáno standardní SPI rozhraní a běžné principy komunikace na této sběrnici, popis využití SPI při ovládání RGB LED pásku a princip funkce hardwarového modulu a způsob konstrukce panelu a jeho napájení el. energií.

2.1 Popis rozhraní SPI

SPI je standardní sériové periferní rozhraní. Používá se běžně k full-duplex synchronní komunikaci mezi řídícími mikroprocesory a dalšími IO (A/D převodníky, digitální displeje, EEPROM paměti,...). Rozhraní je schopné pracovat na poměrně vysokých frekvencích, řádově desítky MHz.

Full-duplex komunikace znamená, že vysílat data na sběrnici můžou v jeden časový interval obě strany zároveň, jednu stranu nazýváme jako SPI Master (nejčastěji tuto stranu zastupuje mikroprocesor) a druhou SPI Slave (display, paměť, atd.).

2.1.1 Zařízení SPI Master

Pokud je zařízení ve funkci SPI Master, znamená to pro něj:

- řídí komunikaci pomocí hodinového signálu (CLK, někdy označován také jako SCLK)
- určuje pomocí Chip Select, se kterým zařízení na sběrnici bude komunikovat v případě, že máme na sběrnici více zařízení

2.1.2 Zařízení SPI Slave

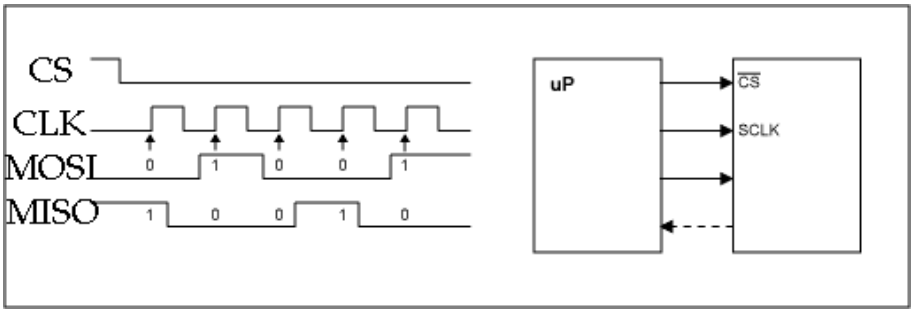
Pokud je zařízení ve funkci SPI Slave, znamená to pro něj:

- podřizuje se straně Master
- vysílá data podle hodinového signálu, pokud je aktivován pomocí výstupu CS

2.1.3 Průběh běžné komunikace na sběrnici

Používají se zde ke komunikaci pouze čtyři vodiče, a to:

- CLK - (Clock) hodinový signál pro komunikaci
- MOSI (Master Out Slave In) - Master zde vysílá data, Slave je přijímá
- MISO (Master In Slave Out) - Master zde přijímá data, Slave je vysílá
- CS (Chip Select, výrobce Raspberry Pi jej nazývá jako CE - Chip Enable) - pro výběr/aktivaci zařízení, se kterým chceme komunikovat



Obrázek 1: Signály SPI sběrnice

Pro začátek komunikace nastaví Master na svém CS výstupu logickou 0 na zařízení, se kterým chce komunikovat, čímž vlastně provede výběr zařízení, pokud je jich připojeno více na jednu sběrnici. Poté začne generovat na výstupu hodinový signál CLK a v té chvíli začnou obě zařízení vysílat svá data, přičemž MOSI působí vždy jako Master výstup, který data vysílá na sběrnici, a Slave je přijímá. MISO je pak právě naopak, tedy Master in, ten data přijímá a Slave out odesílá.

Po vyslání dat zařízení Master může komunikace dále pokračovat:

- buď Master dále dodává hodinový signál a hodnota CS se nemění
- nebo může být komunikace ukončena, tedy Master přestane vysílat hodinový signál a nastaví CS do logické 1

Délka vysílaných dat na sběrnici je typicky 8-bitové slovo. V programovacím jazyku `C` a `C++` můžeme pro toto slovo využít deklarace proměnné jako `char` nebo popřípadě `u_int8`.

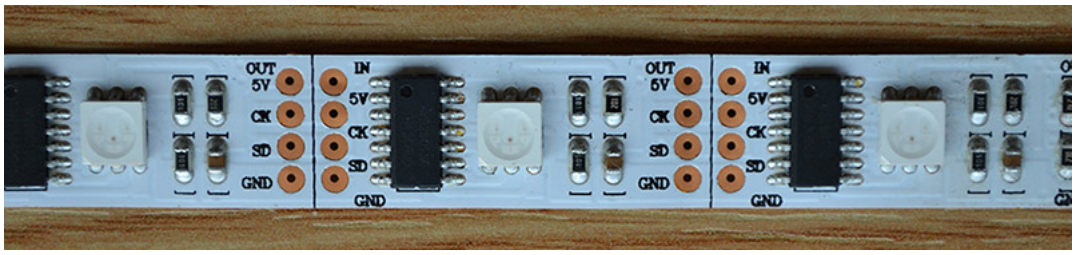
Mimo full-duplex jsou další typy komunikace half-duplex a simplex. Half-duplex znamená, že na sběrnici může data vysílat pouze jedno zařízení, v jeden časový interval, a tedy až po dokončení přenosu může data vysílat druhé zařízení. Komunikace typu simplex znamená, že data vysílá pouze jedno zařízení a ostatní pouze naslouchají a nikdy žádná data nevysílají.

V našem případě budeme používat pouze simplex komunikaci. Raspberry Pi bude data vysílat na sběrnici, ale přijímat již nic nebude potřeba.

Velikou výhodou této sběrnice je její poměrně snadná implementace, a to jak hardwarová, tak i softwarová.

2.2 Popis a ovládání RGB LED pásku

LED pásky, ze kterých je sestaven panel, se prodávají po metrech případně po 5 metrech namotaných na cívce. Na délku jednoho metru pásku vychází 32 pixelů. Pásek tedy lze prodlužovat, nebo zkracovat podle potřeby. Pro zkracování postačí přestřihnout pásek na černé dělící čáře, která je mezi pájecími body. Pro prodloužení pásku je pak potřeba propojit kontakty, tedy prohřát pájecí body a prolít cínem.



Obrázek 2: Pixel LED pásku

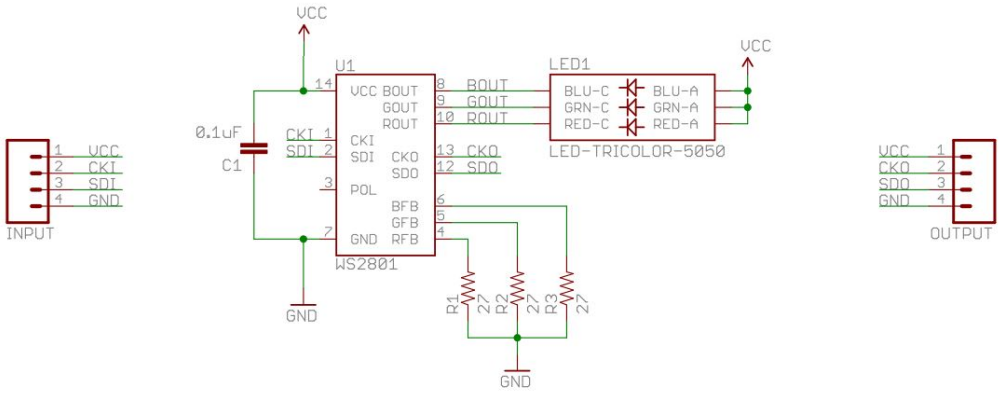
Pájecí body jsou na jedné straně označeny IN, to je vstupní strana. Do vstupní strany vždy musí přicházet na bod CK (Clock) hodinový signál a na SD (Serial Data) datový signál. Je tedy důležité při napojování pásku zachovávat, aby vždy strana označená jako IN byla připojena na stranu označenou OUT, jinak zapojení nebude fungovat. Jednotlivé pájecí body jsou pak označeny jako:

- 5V - slouží k napájení IO a LED a je přechody mezi pájecími body propojen v celé délce pásku
- CK (Clock) - slouží pro vstup hodinového signálu
- SD (Serial Data) - slouží jako datový vstup do pixelu
- GND - slouží jako zem pro napájení a je také propojen v celé délce pásku

Jednotlivé pixely jsou mezi sebou na pásku propojeny sériově. Každý jednotlivý pixel se pak skládá ze šesti součástek:

- Integrovaného obvodu WS2801
- Tříbarevné LED 5050
- 3x rezistorů
- 1x kondenzátoru

A samotné zapojení těchto součástek pak vychází ze schéma navrženého od výrobce IO WS2801. Je zde použita tříbarevná LED 5050 v zapojení se společnou anodou a IO WS2801 pak podle dat, které obdrží na svém vstupu, řídí stmívání jednotlivých složek barev RGB pomocí 3-kanálové PWM (pulzně-šířkové modulace).



Obrázek 3: LED Schéma zapojení pixelu

Integrovaný obvod WS2801 na svůj vstup obdrží z SPI sběrnice tři slova, tedy sekvence logických stavů 0 a 1. Každá tato sekvence má délku osmi bitů, každé slovo tak může nabývat hodnotu 0 až 255. Podle těchto tří slov nastaví IO pomocí svého výstupu intenzitu svitu pro jednotlivé složky RGB barvy. První obdržené slovo je tedy intenzita svitu modré barvy, druhé slovo určuje intenzitu svitu zelené barvy a třetí slovo je pro červenou barvu.

Tyto integrované obvody jsou pak na pásku propojeny sériově. Jakmile první obvod v řadě příjme tři slova, tak na další již nereaguje, pouze je přeposílá dále až do toho okamžiku, dokud se nepřeruší hodinový signál. Po přerušení hodinového signálu zůstává na výstupu IO nastavená intenzita svitu pro pixel, ale integrovaný obvod je opět ve stavu, kdy přijme tři slova a další přeposílá dále.

2.3 Popis panelu

Konstrukce panelu byla uzpůsobena k tomu, aby byl panel bez větších obtíží skladný a přenosný. [1]

2.3.1 Konstrukce panelu

Jako plátno bylo použito silnější linoleum, které je částečně ohebné, ale hlavně dostatečně pevné a odolné. Pro uchycení je na vrchní straně přichycen hliníkový profil, a ten je uchycen na obou koncích šrouby k nastavitelným stojanům na reproduktory, které zajišťují dostatečnou stabilitu.

Na plátno jsou již pak horizontálně přilepeny pásky s RGB LED. Na výšku je na panelu 36 řad pásků a na každém pásku je 64 LED. Ve výsledku máme tedy 2 304 pixelu a rozlišení 64 x 36.

[1] Konstrukce panelu byla dodána vedoucím práce.

Obrázek 4: Panel se stojany

Vzhledem k využití panelu pro zobrazování multimediálních dat byl zvolen dnešní pomyslný standard poměru stran, a to 16:9. Tento poměr byl rozšířen na výšku i na šířku 4 krát a z něj vyplývá rozlišení 64 x 36. Rozteč jednotlivých LED je pak 3 cm v horizontálním i vertikálním směru.

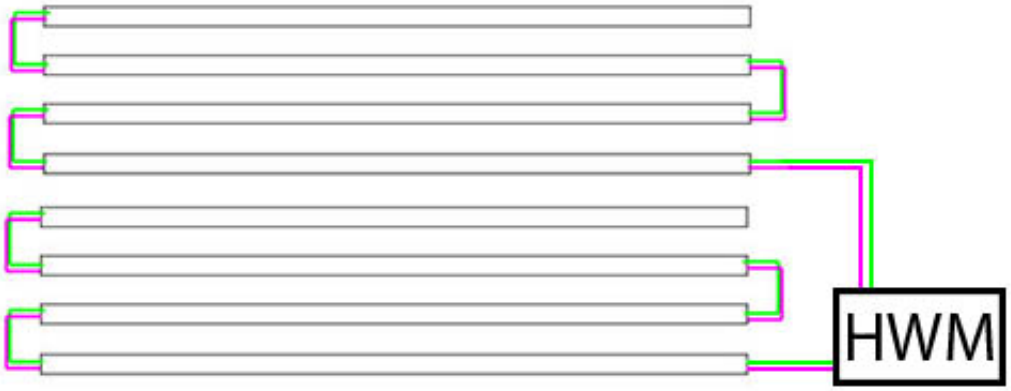
2.3.2 Propojení pásků LED

Poznámka 2.1 V prvním návrhu zapojení bylo tedy všech 36 řad pásků propojeno sériově, zapojení bylo v pořádku a funkční, avšak při nastavení vyšší přenosové rychlosti na sběrnici docházelo k výpadkům u posledních řad. Poslední řady zobrazovaly náhodná nebo žádná data. Docházelo vlivem nedokonalosti součástek a vodičů k fázovému posuvu mezi signály hodin a dat. Řešením tedy bylo ponechat nižší rychlost na sběrnici, a tím tedy zpomalit vykreslování obrazců, což ale není vhodné pro animace. Další možností bylo rozdělit řady na pomyslné segmenty.

Aby na sběrnici bylo dosaženo vyšší přenosové rychlosti, je tedy nutné pásky rozdělit do skupin neboli segmentů. Na plátnu jsou tudíž pásky zapojeny po čtyřech řadách a rozděleny do devíti segmentů. Signálové vodiče ze vstupu dat jednotlivých segmentů jsou pak již připojeny k hardwarovému modulu, který řeší právě výběr segmentu, do něhož se mají data posílat.

K přenosu informace po sběrnici se zde využívají dva vodiče, jeden datový a druhý pro signál hodin. U zapojení jednotlivých segmentů se využívá vlastnosti, že pokud

na pixel přichází pouze data a nikoliv hodinový signál, tak integrovaný obvod WS2801 nereaguje, tedy nemění žádným způsobem svůj výstup. Datový vodič je proto propojen mezi všemi segmenty na panelu a hodinový signál se posílá pomocí hardwarového přepínače pouze do segmentu, který má být překreslen.



Obrázek 5: Ukázka zapojení dvou segmentu a hardwarového modulu

2.4 Hardwarový modul přepínání

Principiální funkčnost modulu pro multiplexované rozhraní SPI byla navržena panem Ing. Davidem Seidlem, Ph,D., my jsme pak s kolegou Josefem Holišem nakreslili schéma zapojení, navrhli desku plošných spojů a následně celou funkci ověřili v praxi na LED panelu.

Multiplexování je název pro proces, kdy se sdružuje více různorodých signálů do jednoho. V tomto zapojení dochází k demultiplexování, tedy rozdělení multiplexovaného hodinového signálu sběrnice pro jednotlivé segmenty panelu.

V navrženém zapojení hardwarového modulu na přepínání vykreslovaného segmentu využíváme část sběrnice SPI. Jde o část z toho důvodu, že panel neprovádí žádnou signalizaci, tudíž odpadá využití signálu MISO (Master In Slave Out). K řízení celého panelu tedy postačí pouze datový výstup (MOSI), hodinový výstup (CLK) a signál Chip Select (CS).

Modul má výhodu, že pro jeho použití není zapotřebí žádný pomocný signál a vystačí si pouze se standardní sběrnici SPI. Pro samotné přepínání segmentů se využívá signálu Chip Select, který při každém zápisu dat na sběrnici vykoná impuls, tedy změní svůj stav při začátku přenosu na logickou 0. Po dokončení komunikace na sběrnici svůj stav nastaví zpátky na logickou 1.

Díky této vlastnosti sběrnice mohl vzniknout jednoduchý modul pro ovládání panelu za pomocí 3 integrovaných obvodu a páru dalších součástek.

2.4.1 Součástky, ze kterých je modul sestaven:

- IC1 - časovač NE555
- IC2 - 4-bitový binární čítač 74HC93
- IC3 - dekodér 1 z 16 74HCT5414N
- tranzistory pro zesílení a invertování vstupních signálů
- další pomocné a kondenzátory, usměrňovací diody a rezistory

Na desce plošných spojů jsou pak k dispozici dva konektory, jeden slouží jako vstupní (označen jako PI, FTDI) a druhý jako výstupní (OUT). Vstupní konektor tedy slouží pro připojení na SPI sběrnici počítače Raspberry Pi. Druhý, výstupní konektor je určen pro připojení LED panelu.

Poznámka 2.2 Během vývoje přibyla mezi LED panel a hardwarový přepínač ještě jedna DPS.[1] Na ní se nachází dva integrované obvody 74HCT541, které mají integrované operační zesilovače v neinvertujícím zapojení. Jejich funkcí je posílení signálu sběrnice a částečná ochrana proti zkratu a možnému vyššímu napětí na panelu nebo na DPS přepínače.

2.4.2 Popis vstupního konektoru modulu

Vstupní konektor se skládá z pěti pinů:

- +5V pro napájení (tento vývod využívám k napájení počítače Raspberry Pi)
- CLK vstup pro hodinový signál sběrnice SPI
- GND pro uzemnění
- CS vstup pro signálu Chip Select sběrnice SPI
- DATA pro vstup dat sběrnice SPI

Poznámka 2.3 Při použití pinu +5V k napájení Raspberry Pi je nutné, aby výstupní napětí na zdrojích bylo nastaveno co nejblíž hodnotě 5V. Pokud je nastaveno nižší napětí, mohou přestat fungovat periferie připojené do USB portu počítače Raspberry Pi. V mém případě tak přestávala reagovat optická USB myš.

[1] Tato DPS byla navržena panem Ing. Petrem Olivkou, Ph.D.

2.4.3 Popis výstupního konektoru modulu

Výstupní konektor se pak skládá z 13 pinů a připojuje se již k ochranné desce plošných spojů na panelu:

- zdvojená zem (GND) pro propojení a vzájemné uzemnění
- +5V pro napájení DPS z panelu
- 9 pinů C8 až C0 jako výstupy hodin pro každý jednotlivý segment
- DATA pro výstup dat na panel

2.4.4 Popis funkce zapojení

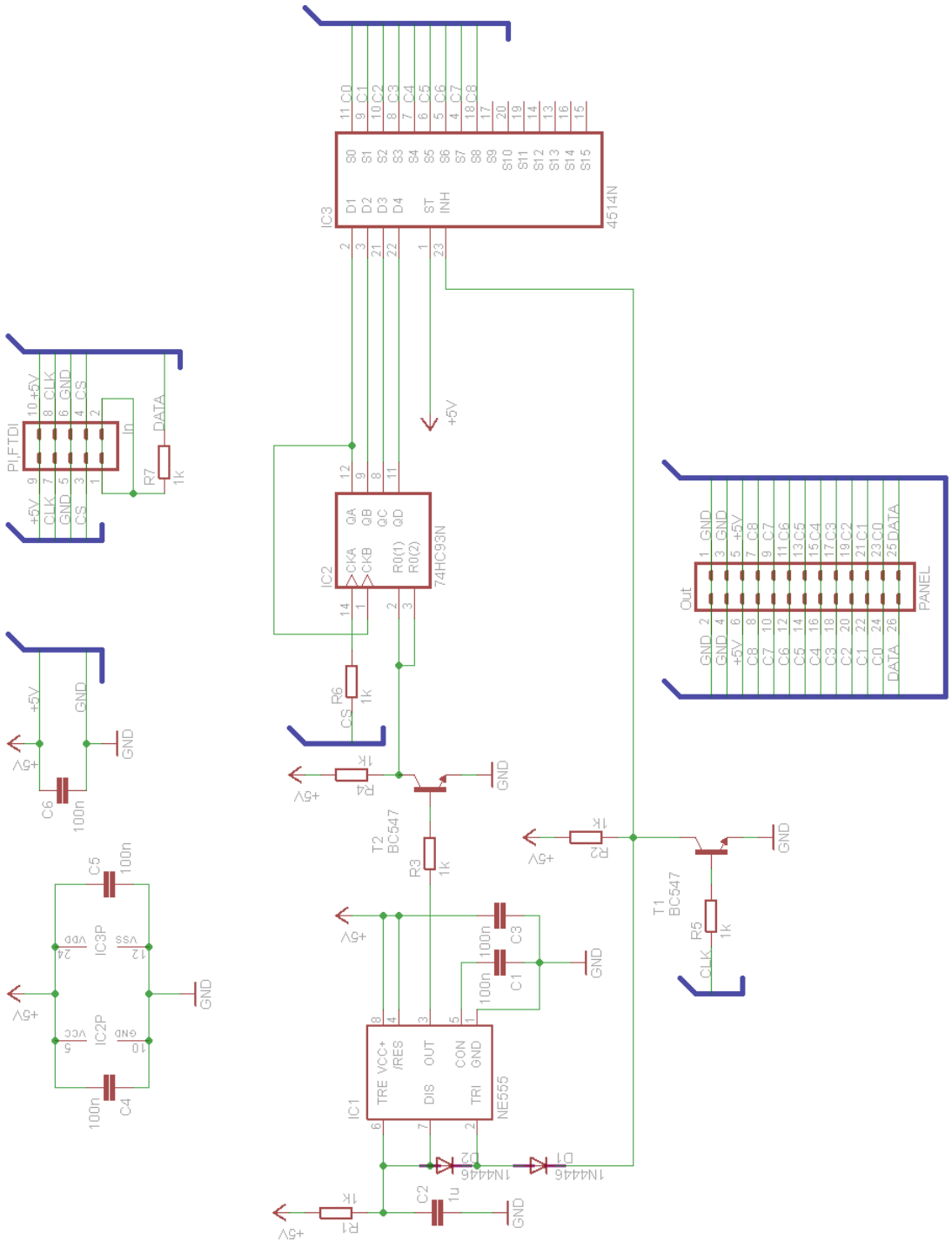
Bezprostředně za vstupním konektorem jsou za piny pro hodinový, datový, a Chip Select signál umístěny rezistory R5, R6 a R7 v rámci jednoduché ochrany. Slouží k ochraně výstupu sběrnice počítače Raspberry Pi před případným zkratem na DPS a následným poškozením, případně zničením výstupu sběrnice SPI na počítači Raspberry Pi.

Vzhledem k tomu, že datový signál sběrnice můžeme posílat do všech segmentů panelu zároveň, nemusíme jej dále řešit a můžeme jej poslat ze vstupního konektoru přímo na výstupní konektor na panel.

Integrovaný obvod IC3 je dekodér 1 z 16. Do dekodéru přivádíme na vstup označen jako IHN invertovaný hodinový signál. Dekodér podle číselné hodnoty na svém vstupu (D1 - D4) posílá hodinový signál dále na odpovídající výstupní pin (S0 - S8). Ostatní výstupy jsou nevyužity, máme pouze 9 segmentů. Hodinový signál je invertován pomocí tranzistoru T1, bezprostředně za vstupním konektorem. Tranzistor T1 je tedy zapojen jako jednoduchý invertor, obrací logiku signálu.

Integrovaný obvod IC2 je 4-bitový binární čítač, umí tedy na svůj výstup nastavit číselnou hodnotu od 0 do 15. Tento čítač má na svůj vstup označený CKA přiveden signál Chip Select ze vstupního konektoru. Počítá počet impulsů, tedy s každým ukončením zápisu na sběrnici, kdy změní signál CS svůj stav, inkrementuje čítač hodnotu o jedničku a nastaví ji na svůj výstup v binární podobě. Nulovací vstup (reset) čítače je připojen přes invertor sestavený z tranzistoru T2 na výstup integrovaného obvodu IC1. Pokud na nulovací vstup přijde logická úroveň 1, tak čítač svůj výstup nastaví na logickou 0.

Integrovaný obvod IC1 je časovač NE555 a je v zapojení jako monostabilní klopný obvod. Má jeden stabilní logický stav, ve kterém může setrvávat libovolně dlouhou dobu. V tomto zapojení je to stav, kdy na svém výstupu (OUT) má hodnotu logické 0 a čítač je ve stavu vynulování. V tomto stavu je kondenzátor C2 plně nabitý přes rezistor R1. V momentu, kdy na sběrnici začne probíhat komunikace, vstupem označeným CLK začne přicházet hodinový signál a kondenzátor se pomocí diody D2, D1 a tranzistoru T1 vybije. Po vybití kondenzátoru C2 dojde k překlopení obvodu do nestabilního stavu, ve kterém setrvává, dokud přichází hodinový signál CLK. Výstup integrovaného obvodu IC1 OUT



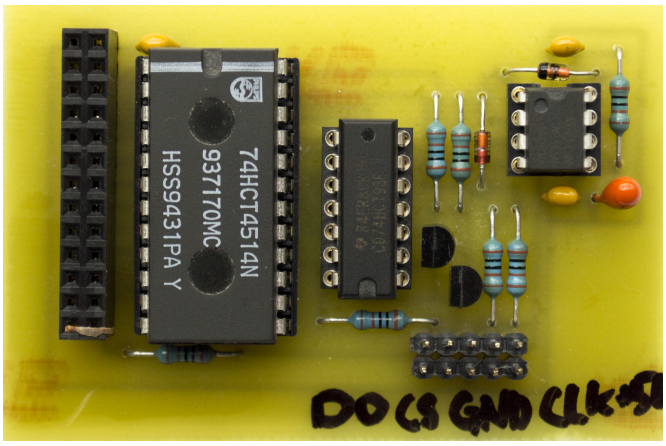
Obrázek 6: Schéma zapojení hardwarového přepínače

je nyní ve stavu logické 1 a čítač je ve stavu, kdy počítá impulsy signálu CS. Pokud přestane na sběrnici probíhat komunikace, tedy přestane přicházet hodinový signál, začne se opět nabíjet kondenzátor C2 pomocí rezistoru R1.

Doba nabíjení kondenzátoru C2 je nastavena pomocí vzorce pro výpočet kombinace kapacity použitého kondenzátoru a velikosti použitého rezistoru R2.

$$t = ln(3) * CR \approx 1,099 * CR \tag{1}$$

V rámci domluvy byla nastavena doba nabíjení kondenzátoru na 1ms. To znamená, že pokud bude mezi komunikací na sběrnici časová mezera více než 1ms, dojde k vynulování čítače a další příchozí hodinový signál půjde opět do prvního segmentu.



Obrázek 7: Fotografie DPS hardwarového přepínače

Poznámka 2.4 Ve standardním zapojení NE555 jako monostabilního klopného obvodu časovač nereaguje na impulsy, které přijdou, pokud je překlopen v nestabilním stavu. Pouze první impuls by obvod překlopil a ihned by se začal nabíjet kondenzátor C2, po nabití by se obvod opět překlopil do stabilního stavu a resetoval by hodnotu na čítači. Z toho důvodu jsme upravili zapojení a to tím, že jsme přidali do zapojení dvě diody D1 a D2.[1] Díky jim nemá kondenzátor dostatečný čas k nabití přes rezistor R1. Dokud přichází hodinový signál CLK a tranzistor T1 obvod uzavírá, klopný obvod setrvává v nestabilním stavu.

2.5 Popis napájení panelu

Při konstrukci LED panelu bylo navrženo napájení pomocí dvou průmyslových spínaných zdrojů z důvodů vyššího odběru elektrického proudu. [2] Napájecí zdroje mají propojené GND výstupy, které vedou z obou zdrojů do všech pásků na panelu kvůli správnému uzemnění.

[1] Spolupráce s kolegou Josefem Holišem

[2] Konstrukce LED panelu včetně napájení byla dodána vedoucím práce.

Liché pásky v celé jejich délce pak napájí jeden zdroj 5 V a sudé pásky napájí zdroj druhý. Toto zapojení bylo zvoleno, aby došlo k rovnoměrnému rozložení zátěže mezi zdroje, které je možné od panelu jednoduše odpojit pro pohodlnější přenášení.

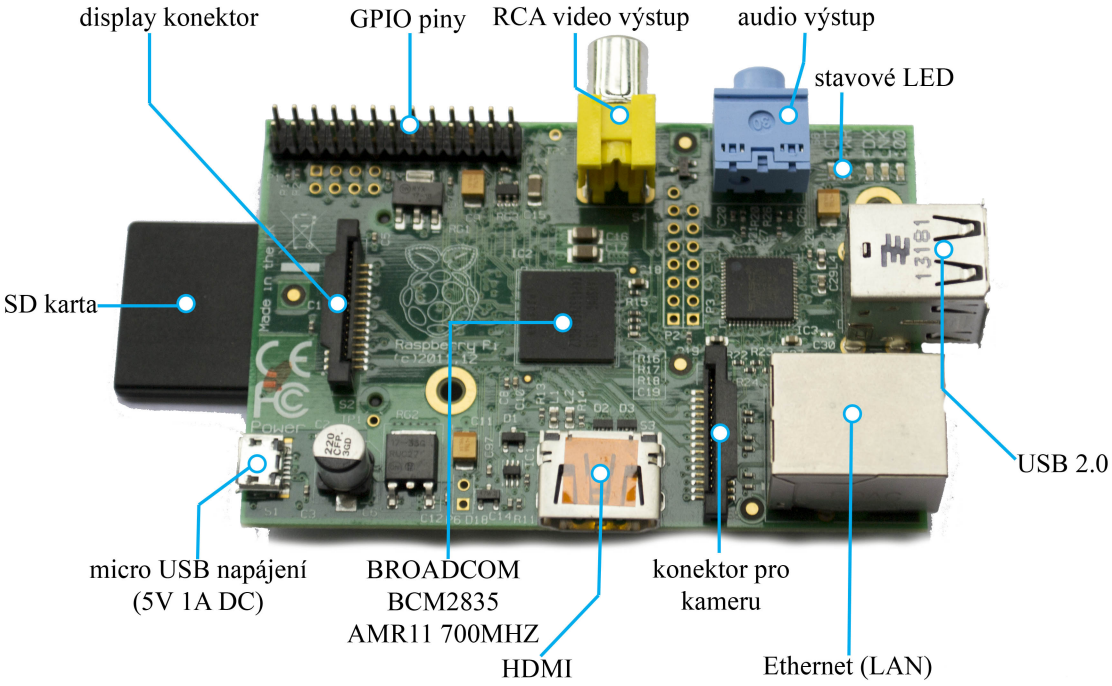
Výkonová řada	320 W
Skutečný výstupní výkon	275 W
Výstupní proud	55 A
Regulace výstupního napětí	4,5..5,5 V
Vstupní napětí AC	88..370 V
Vstupní napětí DC	124..370 V
Účinnost	79 %
Integrované ochrany	Přetížení, zkrat, přepětí, přehřátí
Váha	1,1 Kg
Rozměry	215 x 115 x 50 mm

Tabulka 1: Technické parametry zdrojů

Obrázek 8: Napájení panelu

2.6 Raspberry Pi

Pro ovládání panelu v této práci využívám jednodeskový počítač Raspberry Pi. Jedná se o malý plnohodnotný počítač s DPS velikostí přibližně platební karty. O jeho vývoj se stará britská nadace Raspberry Pi Foundation a tento jejich projekt vznikl s účelem podpory výuky informatiky na školách. Počítač docela rychle získal velikou oblíbenost i u různých kutilů a vývojářů, stal se základem nejrůznějších projektů pro svou relativně nízkou cenu a možnosti instalace operačního systému s mnoha podpůrnými softwary.



Obrázek 9: Raspberry Pi model B rev. 2.0

2.6.1 Specifikace počítače

Základní parametry a specifikace modelu „B“:

- procesor ARM1176JZF-S z rodiny ARM11 taktovaný na frekvenci 700 MHz
- grafický procesor VideoCore IV podporující OpenGL ES 2.0, 1080p30, MPEG-4
- 512 MB RAM sdílených s grafickou kartou
- GPIO port s podporou rozhraní UART, sběrnice I^2C / SPI

- Porty: 2x USB 2.0, HDMI, LAN, audio výstup, RCA video výstup, konektory pro připojení kamery a displaye, slot pro SD kartu a micro USB, který slouží standardně k napájení
- Cena od výrobce je uváděna $35

Aktuálně se počítač vyrábí ve dvou modelech. Model A je levnější verze a je ochuzená o jeden USB port a také o LAN konektor. Dále má pak sníženou velikost paměti RAM na pouhých 256 MB.

Pro mou práci je k dispozici model B rev. 2.0. Na paměťové SD kartě je pak nainstalován operační system Raspbian, který vychází z opensource distribuce Debian pro AMR procesory, a je přímo optimalizován pro Raspberry Pi.

Poznámka 2.5 Pro tento počítač se upravují stále další a nové operační systémy. Jedním z dalších OS je například RaspBMC, který slouží pro použití počítače jako multimediálního centra.

Menší nevýhodou u tohoto počítače je, že kvůli zachování nejnižší možné ceny zde chybí jakákoliv ochrana GPIO portu. Piny jsou vyvedeny přímo z procesoru, a tudíž vzniká riziko nevratného poškození procesoru. Všechna vstupně-výstupní logika GPIO portu je v napětí 3,3V. Pokud by došlo ke zkratu a na port se dostalo vyšší napětí, může ho nenávratně poničit.

3,3V	1	2	5V
I2C 0 SDA	3	4	GND
I2C 0 SCL	5	6	GND
GPIO 4	7	8	UART TXD
GND	9	10	UART RXD
GPIO 17	11	12	GPIO 18
GPIO 21	13	14	GND
GPIO 22	15	16	GPIO 23
GND	17	18	GPIO 24
SPI 0 MOSI	19	20	GND
SPI 0 MISO	21	22	GPIO 25
SPI 0 CLK	23	24	SPI 0 CS0
GND	25	26	SPI 0 CS1

Obrázek 10: Raspberry Pi: popis konektoru GPIO

3 Konfigurace Raspberry Pi a testovací obrazce

3.1 Konfigurace sběrnice SPI na OS Raspbian

Jádro operačního systému Raspbian, tedy kernel ve verzi Linux 3.10.25+ #armv6l, má již plně integrovaný modul pro ovládání SPI sběrnice. Po instalaci systému je však tento modul zakázán a pro funkci SPI sběrnice je nutné nejprve povolit jeho načtení.

Pro povolení modulu za běhu systému slouží program „modprobe“. Pokud modul načteme pomocí tohoto programu, zůstane načtený pouze do restartu systému .

```
$ sudo modprobe spi-bcm2708
```

Výpis 1: Příkaz pro povolení modulu SPI.

Pro povolení automatického načtení modulu při zavádění operačního systému je potřeba upravit konfigurační soubor `/etc/modprobe.d/raspi-blacklist.conf`. Tento soubor obsahuje seznam modulů, které jsou zakázány z důvodu, že je většina uživatelů nevyužije, popřípadě mohou kolidovat s jinou funkcí.

```
$ sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

Výpis 2: Příkaz pro editaci souboru k automatickému zavedení modulu SPI.

V konfiguračním souboru postačí nalézt a zakomentovat řádek s modulem SPI. Po zakomentování bude tedy na řádku napsáno „#blacklist spi-bcm2708“. Nyní je již povoleno zavedení modulu při startu systému.

Po úspěšném zavedení modulu vzniknou ve složce `/dev` dva soubory zařízení, a to `/dev/spidev0.0` a `/dev/spidev0.1`. Při používání `spidev0.0` se bude na GPIO konektoru Raspberry Pi využívat pro výstup pin číslo 24 (SPI 0 CS0) jako signál Chip Select a při použití `spidev0.1` se pak bude používat pin číslo 26 (SPI 0 CS1). Zápis do těchto souborů pak znamená zápis přímo na sběrnici SPI.

3.1.1 První zápis na sběrnici

Pro první test jestli funguje zápis na sběrnici, jsem využil jednoduchého příkazu v konzoli pro zápis do souboru, což je přesměrování výstupu programu „echo“ do souboru zařízení sběrnice SPI.

```
$ echo -ne "\xFF\xFF\xFF" > /dev/spidev0.0
```

Výpis 3: Jednoduchý zápis na sběrnici SPI

Program „echo“ vytiskne string, který dostane na svém standardním vstupu na svůj standardní výstup. Použitý parametr `-n` znamená, že na konci výstupu nebude zalomení řádku, a parametr `-e` znamená, že program bude reagovat na zpětné lomítko ve vstupním textu. Povolí pak tedy zapsat na výstup bajt pomocí hexadecimální hodnoty tedy `xFF`. Tato hodnota je pak jedno slovo zapsané na sběrnici. Na panelu se v okamžiku spuštění tohoto příkazu rozsvítí první pixel bílou barvou.

3.1.2 Testovací program spi.c

Při vývoji ovladače, tedy modulu SPI pro jádro operačního systému, vznikl jednoduchý program v jazyku `C` pro testování a nastavení komunikace na sběrnici SPI.[1] Tento program byl uvolněn jako opensource, tedy s otevřeným zdrojovým kódem a byl publikován pod licencí General Public Licence (GPL). Tímto programem je možné nastavit parametry komunikace, jako jsou například přenosová rychlost na sběrnici, polarita hodinového signálu a další. Kompletní seznam parametrů, které program akceptuje při spuštění, je uveden v tabulce číslo 2.

-D	–device	cesta k zařízení (výchozí: /dev/spidev0.0)
-s	–speed	rychlost sběrnice (Hz)
-d	–delay	zpoždění (μs)
-b	–bpw	počet bitů na slovo
-l	–loop	zpětná smyčka
-H	–CPHA	fáze CLK signálu
-O	–CPOL	polarita CLK signálu
-L	–lsb	bit s nejmenší hodnotou jako první
-C	–cs-high	signál CS je aktivní v log. 1
-3	–3wire	SI/SO sdílený signál

Tabulka 2: Přehled parametrů programu spi.c

Tento program využívám ke změně nastavení vlastností sběrnice. Pokud v něm vlastnost sběrnice nastavím, tak se nastavení promítne do systému a další komunikace na sběrnici již bude probíhat s tímto novým nastavením.

Pro mou práci je podstatné právě nastavení frekvence sběrnice, tedy parametru `-s` na tu hodnotu, kdy data po sběrnici dorazí do všech pixelů na panelu a nebude docházet k chybám zobrazování. Na koncích sériového propojení pixelu v segmentech nedojde k přeslechům nebo fázovému posunu datového signálu a signálu hodin.

Pomocí pokusů jsem ověřil, že navržené zapojení spolehlivě funguje při frekvenci sběrnice 800 kHz.

3.1.3 Výpočet času potřebného pro vykreslení celého panelu

Při zjištění maximální frekvence jsem vypočítal maximální počet vykreslených snímků za vteřinu. Nejdříve za pomocí vzorce a nastavené frekvence sběrnice 800 kHz jsem vypočítal dobu, která je potřebná pro zápis jednoho bitu na sběrnici (t_{bit}).

$$t_{bit} = \frac{1}{f} = \frac{1}{800000} = 0,00000125s = 1,25\mu s$$

[1] Modul jádra a testovací program vyvíjí komunita počítače Raspberry Pi.

Poté jsem vypočítal, kolik bitů je potřeba na sběrnici zapsat. Což znamená nejdříve vypočítat počet pixelů v segmentu, tedy vynásobením šířky (s) a výšky (v_{seg}) segmentu.

$$p_{seg} = s * v_{seg} = 64 * 4 = 256pixel$$

Poté jsem vypočítal počet pixelů na celém panelu (p_{panel}) vynásobením počtu pixelu v segmentu (p_{seg}) počtem segmentů (s) na panelu.

$$p_{panel} = p_{seg} * s = 265 * 9 = 2304pixel$$

Dále stačí vypočítat počet slov a následně počet bitů (p_{bit}) potřebných pro vykreslení celého panelu, tedy počet slov (p_{slov}) vynásobím počtem bitů ve slově (b) a to celé vynásobím počtem pixelů na panelu (p_{panel}).

$$p_{bit} = p_{slov} * b * p_{panel} = 3 * 8 * 2304 = 55296bit$$

Výpočet doby trvání jednoho bitu (t_{bit}) vynásobím počtem bitů (p_{bit}) a získám dobu vykreslení jednoho snímku na celý panel (t_{panel}).

$$t_{panel} = p_{bit} * t_{bit} = 55296 * 0,00000125 = 0,06912s = 69,12ms$$

Po vykreslení celého panelu je potřeba vytvořit zpoždění minimálně 1 ms, aby došlo k vynulování čítače a aby další vykreslovaní začalo opět od prvního segmentu. Musím tedy 1ms přičíst k potřebné době.

$$69,12 + 1 = 70,12ms$$

Po dosazení do vzorce zjistím maximální možný počet snímků vykreslených za vteřinu (FPS).

$$f = \frac{1}{t} = \frac{1}{0,07012} \approx 14,3FPS$$

Teoreticky podle výpočtů mohu zobrazit na panelu při nastavení frekvence sběrnice 800 kHz přibližně 14 snímku za vteřinu.

Prakticky je tato hodnota nižší, při výpočtech totiž dochází k zanedbání doby nutné pro přípravu dat, která je různá podle složitosti úkonu přípravy a dostupného výkonu.

3.2 Testovací obrazce pro LED panel

V rámci ověření zapojení jsem napsal několik jednoduchých programů v programovacím jazyku `C++`. Výhodou tohoto jazyku mimo jiné je také rychlejší vykonávání kódu než u jazyků, jako jsou například `PYTHON` nebo `Java`. Zdrojové kódy jsem kompiloval pomocí kompilátoru `CC`.

3.2.1 Rozsvícení panelu bílou barvou

Jako základní testovací obrazec jsem vybral rozsvícení panelu bílou barvou. Vzhledem k tomu, že u počítače Raspberry Pi je při použití operačního systému Raspbian přístupná sběrnice SPI pomocí souboru zařízení `/dev/spidev0.0`, postačí zapsat data do tohoto souboru.

```
#include <fcntl.h>    // open(), O_RDWR
#include <stdlib.h>   // write(), close()

static int PIXELU_NA_SEGMENT = 4*16*4;
static int POCET_SEGMENTU = 9;

int main(void)
{
   char buffer[PIXELU_NA_SEGMENT * 3];

   for (int d = 0 ; d < PIXELU_NA_SEGMENT * 3 ; d++)
   {
      buffer[d] = 0xFF;
   }
   for (int seg = 0; seg < POCET_SEGMENTU ; seg++ )
   {
      int fd = open("/dev/spidev0.0",O_RDWR);
      write(fd, buffer, PIXELU_NA_SEGMENT * 3);
      close(fd);
   }
return 0;
}
```

Výpis 4: Zdrojový kód programu bila.cpp.

Pro manipulaci se soubory je v jazyku `C++` potřeba načíst hlavičkové soubory fcntl.h a stdlib.h. Po té inicializuji proměnné PIXELU_NA_SEGMENT a POCET_SEGMENTU a nastavím do nich údaje o velikosti LED panelu.

Jako další krok vytvořím pole znaků, jehož velikost získám vynásobením počtu pixelů x 3. Je nutné vynásobit počet pixelů 3x z toho důvodu, že každý pixel potřebuje k jeho řízení právě tři 8-bitová slova.

První cyklus for pak naplní celé pole znaků hodnotami 0xFF. Tato hodnota je hexadecimální zápis decimální hodnoty čísla 255. Každá RGB složka barvy pixelu pak bude svítit maximální intenzitou a výsledná barva pixelu bude pak bílá. Pro nastavení intenzity svitu bílé barvy postačí snížit hodnotu, kterou vkládám do pole buffer.

Druhý cyklus for otevře pomocí funkce `open()` zařízení pro zápis a zapíše pomocí funkce `write()`. Ta přijímá jako své parametry otevřené zařízení, potom buffer, který obsahuje hodnoty připravené pro zápis, a velikost bufferu, který obsahuje data pro zápis. Po dokončení zápisu dojde již pouze k uzavření zařízení pomocí funkce `write()`. Uzavření zápisu zajistí uvolnění sběrnice a signalizuje na ní ukončení komunikace. Tímto je dokončeno vykreslení jednoho segmentu na LED panelu a další průběh cyklu for již vykresluje druhý segment.

3.2.2 Rozsvícení panelu libovolnou barvou

Jako další testovací obrazec jsem použil rozsvícení celého LED panelu libovolnou barvou, kterou nastavím pomocí parametrů při spuštění programu. Zdrojový kód vychází z programu pro vykreslení bílé barvy.

```
#include <stdio.h>    // printf()
#include <sstream>  // write(), close(), istringstream()
...
int main(int argc, char argv[])
...
r = ReadInt(argv[1]);
g = ReadInt(argv[2]);
b = ReadInt(argv[3]);
...
for (int d = 0 ; d < PIXELU_NA_SEGMENT * 3 ; d++)
   {
      buffer[d++] = r;   // (R)
      buffer[d++] = g;   // (G)
      buffer[d] = b;     // (B)
   }
...
```

Výpis 5: Část kódu programu barva.cpp

V tomto programu je nutné načíst hlavičkové soubory stdio.h a sstream pro použití funkcí, které umožňují komunikaci programu s uživatelem.

Pro převod vstupních parametrů programu na datový typ integer jsem použil funkci `istringstream()`, která přijímá jako parametr řetězec znaků. Pokud se povede převod znaků na integer, tak navrátí číselnou hodnotu. Pokud převod selže, funkce navrátí hodnotu 0.

```
int ReadInt(char* arg)
{
   int ret;
   istringstream ss(arg);
   if (!(ss >> ret))
      printf ("Chyba: Zadanou hodnotu '%s' nelze prevest na cislo. \n", arg);
   if (ret > 256)
   {
      printf ("Chyba: Hodnota '%d' je prilis velika, zadejte cislo od 0 do 255\n", ret);
      exit(1);
   }
   return ret;
}
```

Výpis 6: Funkce ReadInt()

Moje funkce `ReadInt()` převede textovou hodnotu, kterou jí předám jako parametr na integer. Pokud je výsledná hodnota čísla vyšší než 256, program vypíše chybovou hlášku a ukončí se s chybovým stavem. V případě, že hodnotu nelze převést na datový typ integer, navrátí funkce číslo 0 a vypíše chybovou hlášku.

3.2.3 Vykreslení jednotlivých segmentů

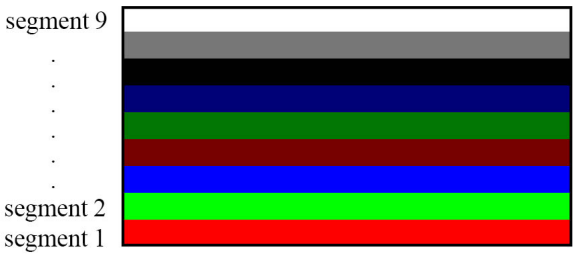
Pro vykreslení každého segmentu jinou barvou jsem upravil zdrojový kód programu bila.cpp.

První cyklus prochází segmenty, v druhém cyklu pak již pouze nastavuji intenzitu svitu RGB barev pro pixely v aktuálně vykreslovaném segmentu.

```
...
  for (int seg = 0; seg < POCET_SEGMENTU ; seg++ )
  {
    for (int d = 0 ; d < PIXELU_NA_SEGMENT * 3 ; d++)
    {
      if (seg == 0)  //pokud je segment 1
      {
        buffer[d++] = 0x00;      //(B)
        buffer[d++] = 0x00;      //(G)
        buffer[d] = 0xFF;      //(R)
      }
      if (seg == 1)  // pro segment 2
      {
        buffer[d++] = 0x00;      //(B)
        buffer[d++] = 0xFF;      //(G)
        buffer[d] = 0x00;      //(R)
      }
       ...
      if (seg == 8)  // pro segment 9
      {
        ...
      }
    }
    int fd = open("/dev/spidev0.0",O_RDWR);
    write(fd, buffer, PIXELU_NA_SEGMENT * 3);
    close(fd);
  }
...
```

Výpis 7: Část kódu programu segmenty.cpp

Volba jednotlivých segmentů zde probíhá pomocí výše uvedených podmínek. Po spuštění tohoto programu se každý segment LED panelu rozsvítí jinou barvou.



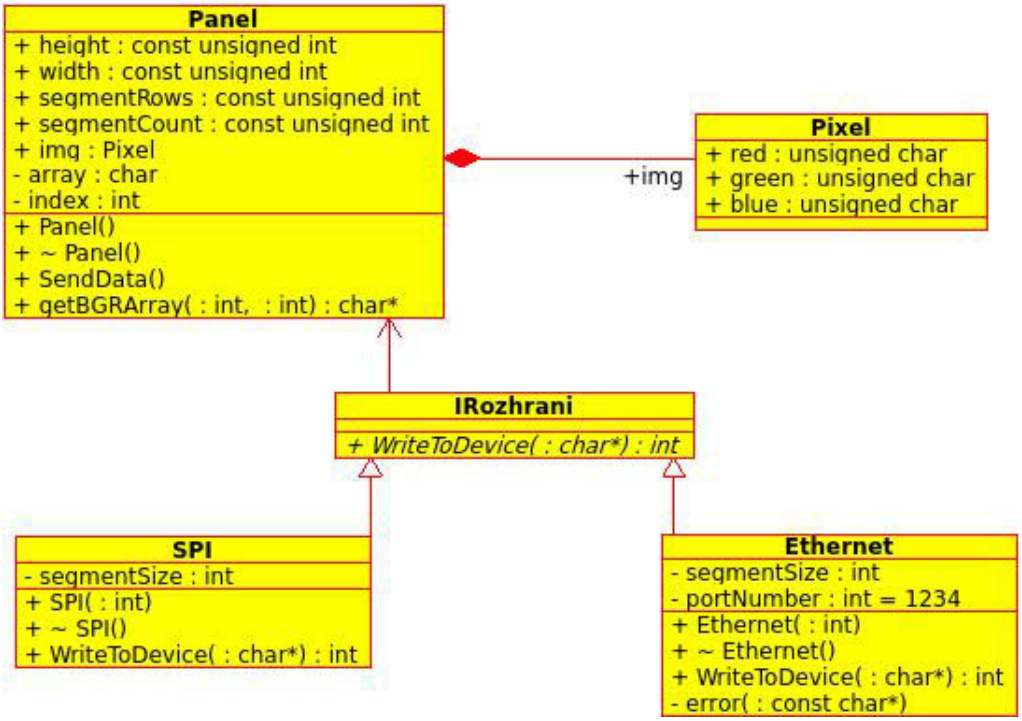
Obrázek 11: Vykreslení jednotlivých segmentů programem segmenty.cpp

4 Rozhraní pro zobrazování multimediálních dat

V této kapitole popíši knihovnu tříd, kterou jsem vytvořil za účelem přípravy dat pro vykreslení na LED panel. Tuto knihovnu poté využiji při vykreslování multimediálních dat.

4.1 Knihovna tříd Panel

Knihovnu jsem napsal tak, aby bylo možné její využití i s jinými rozměry panelu a případně i s jiným rozhraním pro připojení panelu.



Obrázek 12: Diagram tříd knihovny Panel

4.1.1 Třída Panel

Tato třída představuje LED panel, má tedy jeho základní parametry jako své statické atributy:

- height - počet řad na výšku
- width - počet pixelů na řádku
- segmentRows - počet řádků v 1 segmentu
- segmentCout - počet segmentů vypočítaný z předešlých hodnot atributů

Datová struktura Pixel reprezentuje jeden pixel na LED panelu. Uchovává v sobě uloženy hodnoty červené, zelené a modré barvy s číselnou hodnotou 0 - 255, kterou reprezentuje datový typ unsigned char.

```
struct Pixel
{
    unsigned char red;
    unsigned char green;
    unsigned char blue;
};
```

Výpis 8: Datová struktura Pixel představující jeden LED pixel panelu

Dalším atributem je dvourozměrné pole struktury Pixel pojmenované img. Pole má stejné rozměry jako je rozlišení LED panelu. Toto pole slouží pro vkládání hodnot jednotlivých barev každého pixelu z obrázku, který má již rozměry shodné, jako je rozlišení LED panelu.

Jako privátní atribut je definováno pole array, které slouží jako odkládací prostor pro hodnoty, které jsou již připraveny přímo pro zápis na sběrnici. Velikost pole je rovna množství zapisovaných dat.

4.1.1.1 Metoda getRGBArray(int,int)

Tato metoda přijímá 2 parametry. První je číslo, které ukazuje segment, pro který má metoda vrátit pole dat. Druhým parametrem je počet řádků na jeden segment. Pole dat, které funkce vrátí, je pak připravené pro odeslání přímo na sběrnici SPI. Použité proměnné height a width jsou třídní konstanty, které určují rozměry panelu.

```
char* Panel::getBGRArray(int segment, int segRows)
{
    index = 0;
    int from = segment * segRows;
    int to = from + segRows;
   for (int h = from; h < to; h++)      {
      int row = height - h - 1; // begin from bottom
      for (int w = 0; w < width; w++)
      {
          if (row % 2)
          {
              int coll = width - w - 1; // read pixels from right
              array[index++] = img[row][coll].blue;
              array[index++] = img[row][coll].green;
              array[index++] = img[row][coll].red;
          }
          else
          {
              int coll = w; // read pixels from left
              array[index++] = img[row][coll].blue;
              array[index++] = img[row][coll].green;
              array[index++] = img[row][coll].red;
          }
      }
```

```
    }
    return array;
}
```

Výpis 9: Metoda getBGRarray(int)

V tomto mém algoritmu dojde nejdříve k výpočtu, na kterém řádku začínají data pro segment. Dále se vypočítá, kterým řádkem segment končí. První cyklus for začne procházet řádky vykreslovaného segmentu. Proměnná row je přepočet, který zajišťuje, že řádky patřící do segmentu se budou procházet od spodního.

Druhy cyklus for zajišťuje převod dat ze struktury dvourozměrného pole Pixelu do jednoduchého pole, které již obsahuje hodnoty ve správném formátu pro zapsání do segmentu LED panelu. Z konstrukce panelu vyplývá, že je nutné sudé řádky vykreslovat z levé strany a liché řádky ze strany pravé. Tento problém jsem vyřešil pomocí konstrukce podmínek.

4.1.1.2 Metoda Draw()

Metoda slouží k vykreslení dat na LED panel. Za pomoci rozhraní pojmenovaného IRozhrani vytvoří již přímo instanci třídy, která má na starosti zápis dat do panelu.

```
void Panel::Draw()
{
    #ifdef LAN
        IRozhrani *lan_dev = new Ethernet(segmentRows * segmentCount * width * 3);
        lan_dev->WriteToDevice(getBGRArray( 0, segmentRows * segmentCount ));
        delete lan_dev;
    #else
        IRozhrani *spi_dev = new SPI(segmentRows * width * 3);
        for (int i = 0 ; i < segmentCount ; i++ )
        {
            spi_dev->WriteToDevice(getBGRArray(i,segmentRows));
        }
        delete spi_dev;
    #endif
}
```

Výpis 10: Metoda Draw()

V této metodě jsem za pomocí makra vytvořil 2 způsoby zápisu dat do panelu. Pokud při kompilaci zdrojového kódu přidám identifikátor `-DLAN`, tak preprocesor připraví kód tak, že se bude kompilovat blok kódu, který je označen #ifdef LAN. Pokud preprocesor nenalezne identifikátor `-DLAN`, tak se bude kompilovat kód, který je v bloku #else.

Třídy IRozhraní, SPI, a Ethernet a jejich metody popíši dále.

4.1.2 Třída IRozhrani

Tato třída je pouze rozhraní a má jednu virtuální metodu `WriteToDevice(char*)`, kterou pak implementují třídy SPI a Ethernet.

4.1.3 Třída SPI

Tato třída se stará o zápis dat do LED panelu, má konstruktor s jedním parametrem, který určuje počet bajtů k zápisu. Obsahuje implementaci metody `WriteToDevice(char*)`. Algoritmus této metody je primitivní zápis do souboru, tímto zápisem pak už pomocí modulu jádra operačního systému probíhá zápis přímo na sběrnici.

```
void SPI::WriteToDevice(char* pole)
{
    int fd = open("/dev/spidev0.0",O_RDWR);
    write(fd,pole,segmentSize);
    close(fd);
}
```

Výpis 11: výpis funkce SPI::WriteToDevice()

4.1.4 Třída Ethernet

Tato třída slouží pro odeslání balíku dat určenych pro zobrazení na LED panelu pomocí UDP protokolu přes síťové rozhraní. Slouží k tomu standardní soketové API, které používá Linux. Data se neodesílají po jednotlivých segmentech, ale jako jeden balík, proto také do konstruktoru předávám celkový počet bajtů pro celý panel.

Tuto třídu jsem naprogramoval, aby bylo možné data připravit na jiném, výkonnějším počítači. Na počítači Raspberry Pi je spuštěna moje serverová aplikace, která čeká na příchozí data. Jakmile data po síti dorazí, aplikace je pouze rozdělí a zapíše do jednotlivých segmentů LED panelu.

Při přenosu po síti nejsou data šifrována, nedochází k ověření jejich odesílatele a ani se neověřuje doručení. Čekání na ověření přijetí dat totiž způsobovalo další zpomalení přenosu a pokles snímků vykreslených za vteřinu.

Pro použití této třídy jsem k počítači Raspberry Pi do USB portu připojil USB WiFi síťový modul a nakonfiguroval jsem ho jako přístupový bod (AP). Nastavil jsem na počítači Raspberry Pi na rozhraní wlan0 statickou IP adresu. Pro použití modulu v režimu AP jsem do systému Raspbian nainstaloval aplikaci hostapd a nakonfiguroval parametry vysílané sítě. Název vysílané sítě(SSID) je „Panel“ a šifrování jsem nastavil na WPA2 PSK a dále heslo: „malinamalina“

Pro přidělování IP adres počítačům, připojujícím se k této WiFi síti jsem použil aplikaci udhcpd(DHCP server), která je tak jako hostapd dostupná v repozitáři.

IP adresu počítače Raspberry Pi na rozhraní wlan0 jsem pak nastavil na 192.168.0.1/24 a server naslouchá na portu 1234. Počítač, který se připojí k bezdrátové síti Panel, obdrží od DHCP serveru IP adresu z rozsahu 192.168.0.20 - 192.168.0.254.

4.2 Zobrazení statického obrazu

Pro zobrazování statických obrázků je potřeba, aby měl obrázek stejné rozlišení jako LED panel, případně je nutné pomocí funkcí upravit jeho rozměry. Pokud již má správné rozlišení, postačí z obrázku vyčíst hodnoty barev pro jednotlivé souřadnice.

4.2.1 Knihovna OpenCV

Tato knihovna (Open Computer Vision) je projekt s otevřeným zdrojovým kódem a je volně dostupná pro akademické i komerční využiti. Knihovna je napsaná v programovacím jazyku C++, ale obsahuje rozhraní i pro jazyky Python, Java, MATLAB. Podporuje operační systémy Windows, Linux, Androi, Mac OS a zaměřuje se na grafické operace v reálném čase.

Po instalaci knihovny v operačním systému Raspbian je možné knihovnu začít využívat. Pro použití metod z knihovny je pak nutné načíst hlavičkové soubory knihovny.

```
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
```

Výpis 12: Hlavičkové soubory OpenCV pro jazyk C++

4.2.2 Program cvled.cpp

Tento program využívá knihovnu Open CV a slouží k vykreslení obrázku, který je ve formátu *.bmp, *.jpg, *.jpeg, *.png nebo *.tif, na LED panel. Při spouštění programu předávám jako parametr cestu k obrázku, který potřebuji vykreslit.

```
$ ./cvled kostka.png
```

Výpis 13: Spuštění programu cvled.cpp

V tomto programu již využívám i mou knihovnu Panel, a je tedy nutné ji pomocí #include načíst.

```
...
#include "Panel/panel.h"

Panel *vystup = new Panel();
CvScalar s;

int main(int argc, char *argv[])
{
    IplImage *image, *simage;
    image = cvLoadImage(argv[1]);
    if (!image) printf("Picture load: fail \n");
    simage = cvCreateImage(cvSize(vystup->width,vystup->height),image->depth,image->
        nChannels);
    cvResize(image,simage,CV_INTER_LANCZOS4) ;
    for (int y=0 ; y< vystup->height ; y++ )
    {
        for (int x=0; x < vystup->width ; x++ )
        {
            s=cvGet2D(simage,y,x);
            vystup->img[y][x].blue = (char)s.val[0];
            vystup->img[y][x].green = (char)s.val[1];
            vystup->img[y][x].red = (char)s.val[2];
            }
```

```
    }
    vystup->Draw();
    delete vystup;
    return 0;
}
```

Výpis 14: Část zdrojového kódu programu cvled.cpp

Nejdříve vytvořím instanci třídy Panel pojmenovanou „vystup“ a inicializuji proměnnou „s“ s datovým typem CvScalar, tento datový typ pochází z knihovny OpenCV a představuje jeden pixel na obrázku.

Proměnné image a simage jsou datového typu IplImage. Do proměnné image se načte pomocí funkce cvLoadImage() obrázek ze vstupního souboru. Proměnná simage je prázdný obrázek o rozlišení shodném s LED panelem, s barevnou hloubkou a počtem kanálů shodným s vstupním obrázkem.

Metoda cvResize() slouží k transformování velikosti obrázku, jako své parametry přijímá původní načtený obrázek , druhý obrázek do kterého se má provést transformace a algoritmus pro transformaci. Jako tento algoritmus využívám již implementovanou metodu Lanczosovy interpolace pomocí pixelů v okolí 8 x 8 (CV_INTER_LANCZOS4)[1].

Jako další krok již jen pomocí dvou cyklů procházím celý transformovaný obraz a vyčítám hodnoty barev pro jednotlivé pixely do proměnné, a to s pomocí funkce cvGet2D(). Po načtení hodnot jednoho pixelu, již stačí přetypovat a překopírovat hodnoty do odpovídajícího pixelu v atributu img objektu „vystup“.

Poté, co atribut img objektu „vystup“ obsahuje hodnoty pro každý pixel, zavolá se na objekt „vystup“ metoda Draw() a dojde k vykreslení připravených dat na LED panel.

4.3 Dynamické zobrazení pracovní plochy

Pro zobrazování ostatních multimediálních formátů jsem zvolil možnost zrcadlení pracovní plochy na LED panel.

Snímek obrazovky je možné získat pomocí různých knihoven, určenych pro práci s grafickým rozhraním.

4.3.1 Vykreslení plochy počítače Raspberry Pi

V operačním systému Raspbian lze pro získání obrázku obrazovky počítače využít Linux Framebuffer. Jedná se o nejnižší grafickou, hardwarově nezávislou abstraktní vrstvu pro vykreslování počítačové grafiky na této obrazovce. Slovem framebuffer(fb) je tedy myšlena část videopaměti, která obsahuje vždy aktuální obraz.

Nejprve je tedy důležité otevřít framebuffer a získat z něj pevné a proměnné informace o rozlišení vykreslované obrazovky, dále počet bitů na pixel, atd.

```
// path to fb dev
char fbdevice[] = "/dev/fb0";
```

[1] Využití této metody mi bylo doporučeno odbornými grafiky ve škole

```
// open fb dev
int fbfd = open(fbdevice, O_RDWR);

// Get fix info from fb
struct fb_fix_screeninfo finfo;
ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)

// Get variable info from fb
struct fb_var_screeninfo vinfo;
ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)
```

Výpis 15: Část kódu pro získání informací z framebufferu

Všechny tyto funkce jsou ve výsledném kódu ochráněny proti chybě čtení a přístupu. Do výpisu jsem dal pouze podstatné konstrukce.

Když mám k dispozici informace o fb, mohu již vypočítat masky pro čtení jednotlivých barev a počet bajtů na pixel. Vytvořim s pomocí knihovny OpenCV prázdný obrázek o velikosti rozlišení obrazovky (image) a druhý obrázek o velikosti rozlišení panelu (simage).

V dalším kroku již namapuji videopaměť funkcí mmap() a pomocí dvou cyklů, masky pro každou barvu a bitového posunu po namapované části paměti, vyčtu hodnoty barev pro jednotlivé pixely, zapíši je pomocí funkce cvSet2D() do prázdného obrázku (image). Po překopírování hodnot odstraním namapovanou část paměti a provedu pomocí funkce cvRezize() transformaci obrazu s rozlišením obrazovky (image) na obraz s rozlišením panelu (simage).

Na závěr hodnoty překopíruji hodnoty zmenšeného obrazu do atributu img objektu „vystup“ a zavolám metodu objektu Draw(). Po vykreslení uspím proces na dobu 50 μs a poté namapuji nový snímek z framebufferu a pokračuji namapováním nového snímku do paměti a dalšími výše zmíněnými kroky v nekonečné smyčce. Program lze ukončit stiskem kontrolních kláves CTRL + C.

4.3.2 Vykreslení plochy vzdáleného počítače pomocí knihovny Xlib

Pro zobrazení plochy vzdáleného počítače jsem použil sadu knihoven Xlib, která implementuje protokoly pro komunikaci mezi aplikacemi (klienty) a X serverem v grafickém prostředí (GUI).

V programu screen_x11.cpp používám tuto knihovnu k získání aktuálního snímku obrazovky z X serveru operačního systému. Nejdříve inicializuji proměnnou display a získám informace o rozlišení obrazovky. Poté za pomoci knihovny OpenCV vytvořím prázdný obrázek s rozlišením displaye (image) a druhý obrázek opět s rozlišením panelu (simage).

V hlavní smyčce programu pak zašlu pomocí knihovny Xlib požadavek pro získání snímku obrazovky. Z něj pomocí masky pro každou barvu a dvou cyklů for, které procházejí všechny pixely snímku obrazovky, vyčtu hodnoty jednotlivých barev pro každý pixel. Hodnoty pak již opět pouze uložím do prázdného obrázku (image) a za pomoci knihovny OpenCV jej transformuji do rozměrů obrázku simage.

Hodnoty zmenšeného obrázku (simage), překopíruji pomocí 2 cyklů for do atributu img objektu „vystup“ a zavolám na objekt „vystup“ metodu Draw();

Tento program jsem napsal hlavně pro využití zobrazování plochy vzdáleného počítače, a tedy pro použití třídy Ethernet k zaslání balíku dat na LED panel po LAN síti. Do kódu jsem přidal měření času, jak dlouho trvá příprava dat a odeslání po síti LAN.

Základní parametry počítače, který jsem využíval k odesílání data pro panel:

- procesor: Intel Core2 Duo T5870 taktovaný na 2.00 GHz
- grafická karta: integrovaná Mobile Intel GM45 Express Chipset
- rozlišení displaye: 1200 x 800 pixelů
- OS: Linux Ubuntu 14.04 LTS

Počítač s těmito parametry zvládá přípravu a odeslání dat průměrně za 150 ms. Vykreslení panelu mou serverovou aplikací probíhá rychleji, není nutné přidávat zpoždění před další odeslání dat.

Výsledná obnovovací frekvence panelu za použití výše uvedeného počítače je průměrně okolo 7 snímku za vteřinu (FPS).

5 Animace určené k propagaci projektu

Naprogramoval jsem 2 animace pro propagaci projektu. Další animace je pak ve formátu videa uložena na CD.

5.1 Animace Had

Animace zobrazuje barevné „hady“, kteří putují po řádcích LED panelu a mění svou barvu. Program využívá změny parametru barvy, a to odstínu, který je technicky definován v CIECAM02 barevném modelu. Tento model definuje pro vykreslování barev vlastnosti jako jas, barevnost, sytost, odstín a další. Vlastnost odstín má pak základní barvy červenou, zelenou, modrou a žlutou.

Obrázek 13: Kolo odstínů (HUE)

Každý had, který putuje po panelu, mění svou barvu podle kola odstínu (HUE).

Tento algoritmus jsem převzal z jiného projektu, přepsal jsem jej do programovacího jazyka C a upravil jsem jej pro své použití. Zdrojový kód programu je na CD pod názvem had.c

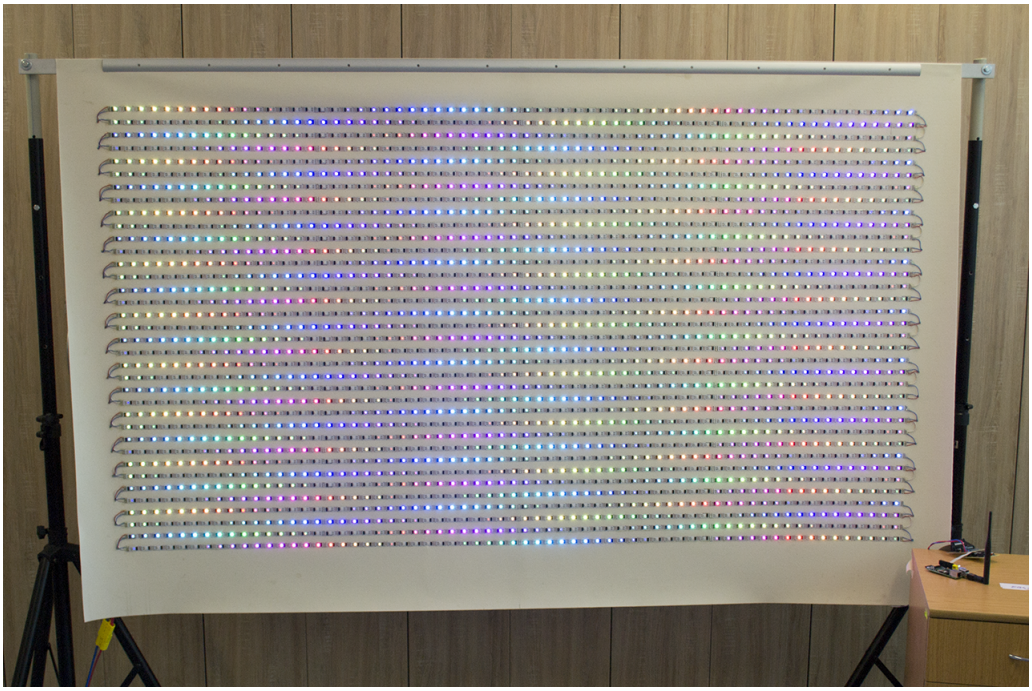
5.2 Animace Hodiny

Tato animace vykresluje na LED panel analogové hodiny, které jsou synchronizovány se systémovým časem. Pro vykreslení hodin využívám funkce knihovny OpenCV pro kreslení do obrázku.

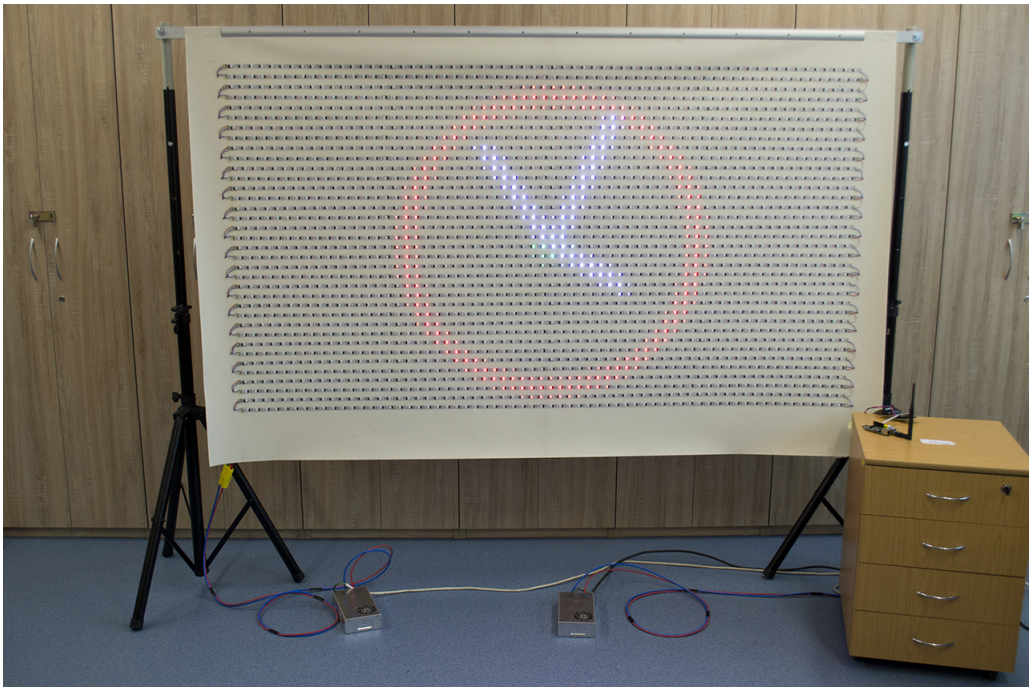
Nejdříve vykreslím do obrázku imgClk kružnice, které budou tvořit ohraničení hodin, pomocí funkce circle(). Po vykreslení obrázek duplikuji do proměnné faceBackUp, abych nemusel pokaždé znovu vykreslovat kružnice.

V dalším kroku získám systémový čas a převedu jej na úhly pro vykreslení úseček, které budou představovat tři ručičky hodin. Poté vypočítám pomocí funkci sin() a cosin() souřadnice jednotlivých úseček a ty pomocí funkce line() pro každou ručičku vykreslím.

Poté výsledek zobrazím na obrazovce a opět pomocí 2 cyklů for projdu všechny pixely obrázku imgClk a překopíruji je do struktury img objektu „vystup“ a na objekt samotný zavolám jeho metodu Draw(); Vymažu obrázek v proměnné imgClk a duplikuji do něj obrázek faceBackUp, který obsahuje pouze vykreslené kružnice. Běh programu lze přerušit pomocí klávesy Esc.



Obrázek 14: Animace Had



Obrázek 15: Analogové hodiny

6 Závěr

Prvním krokem mé práce bylo detailně popsat konstrukci LED panelu, principy sběrnice SPI a princip ovládání LED pásků. S kolegou Josefem Holišem, který pracuje na ovládání LED panelu pomocí USB FTDI kabelu, jsme vyrobili hardwarový modul pro multiplexované rozhraní SPI, který je nezbytně nutný pro další vývoj této práce.

Hlavní podstatou bylo vytvoření knihovny tříd Panel, která usnadňuje programátorům použití LED panelu a umožňuje napsat vlastní rozhraní pro zápis dat na LED panel.

Dále pak jsem vyvinul sérii aplikací, které umožňují na LED panelu zobrazovat základní testovací obrazce, obrázky uložené v souborech typu *.jpg, *.png a dalších. Pro univerzálnější využití jsem napsal aplikaci pro zobrazovaní plochy počítače Raspberry Pi a pro zobrazování plochy ze vzdáleného počítače.

V rámci dalšího vývoje tohoto projektu by mohlo dojít například ke kalibraci barev, konkrétně při zobrazování obrazců, které obsahují světlá místa. Tato místa jsou pak přesvětlena a ruší celkový dojem vykreslovaného obrazce.

Vývoj probíhal na počítači Raspberry Pi a také na notebooku s operačním systémem Linux Ubuntu 14.04 LTS.

Při práci na tomto projektu jsem si připomněl návrh desky plošných spojů a rozšířil své znalosti programovacích jazyků C a C++. Tato práce byla pro mne velmi zajímavá a v mnoha ohledech přínosná.

Tomáš Štrbačka

7 Reference

[1] Luděk SKOČOVSKÝ, a Scott JERNIGAN. Linux: dokumentační projekt. 4., aktualiz. vyd. Překlad Ivo Fořt, David Krásenský. Brno: Computer Press, 2007, 1334 s. ISBN 978-80-251-1525-1

[2] Eben UPTON, a Gareth HALFACREE. Raspberry Pi user guide. 2. aktualiz. vyd. Chichester, England: John Wiley, c2012. xiv, 248 p. ISBN 11-184-6446-X

[3] RASPBERRY PI FOUNDATION. Raspberry PI [online]. Dostupné z: https://www.raspberrypi.org

[4] COMUNITY. Raspbian [online]. Dostupné z: http://www.raspbian.org/

[5] Extérní sériové sběrnice SPI a I2C. [online]. 2008. Dostupné z: http://www.root.cz/clanky/externi-seriove-sbernice-spi-a-i2c/

[6] Sockets Tutorial C. [online]. Dostupné z: http://www.linuxhowtos.org/C_C++/socket.htm

[7] Texas Instruments [online] SN74HC4514NT [Datasheet (PDF)]. Dostupné z: http://pdf1.alldatasheet.com/datasheet-pdf/view/546155/TI/SN74HC4514NT.html

[8] NXP [online] 74HC541 [Datasheet (PDF)]. Dostupné z: http://www.nxp.com/documents/data_sheet/74HC_HCT541.pdf

[9] Circuit simulator. Dostupné z: http://www.falstad.com/circuit/

[10] Worldsemi [online] WS2801 [Datasheet (PDF)] Dostupné z: http://www.adafruit.com/datasheets/WS2801.pdf

[11] Breakout Board WS2801 v11 [online] Dostupné z: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/BreakoutBoards/WS2801-Breakout-v11.pdf

[12] Spínaný zdroj MEAN WELL SP-320-5 [online] Dostupné z: http://www.gme.cz/spinany-zdroj-mean-well-sp-320-5-p332-358

8 Přílohy

K práci je přiloženo CD které obsahuje:

- Složku AP která obsahuje konfigurační soubory pro použití WiFi modulu
- Složku img obsahující testovací obrázek pro použití programu cvled
- Složku src která obsahuje zdrojové kódy programů bílá, barva, segmenty, cvled, screen_fb, screen_x11, server, had, hodiny a ve složce Panel zdrojové kódy knihovny tříd Panel
- Složku video která obsahuje krátkou animaci ve video formátu, která slouží k propagaci LED panelu